Bab 5

Memasukkan dan Memanipulasi Data

Setelah di Bab 4 diterangkan cara membuat tabel, maka sekarang kita akan menggunakan tabel tersebut yaitu mengisinya dengan data.

5.1 INSERT INTO

```
-- Resep 5-1-1: Sintaks dasar perintah INSERT

-- tanpa nama-nama kolom
INSERT INTO nama_table VALUES (ekspresi|DEFAULT[, ...])

-- dengan nama-nama kolom
INSERT INTO nama_table (nama_kolom      [, ...])
                VALUES (ekspresi|DEFAULT[, ...])
```

Perintah SQL INSERT adalah cara utama di SQL untuk memasukkan baris baru ke dalam sebuah tabel. Ada 2 varian utama cara menulis perintah ini, yang pertama dengan menyebutkan nama kolom dan yang kedua tanpa menyebutkan nama kolom. Jika kita tidak menyebutkan nama kolom, maka kita harus mengisi nilai semua kolom dan dengan urutan yang sama sewaktu kita membuat tabel dengan CREATE TABLE. Jika kita menyebutkan nama kolom, maka kita bisa hanya menyebutkan kolom-kolom tertentu saja yang kita kehendaki dan sisanya akan diset dengan nilai default kolom yang bersangkutan.

Tentu saja, perintah INSERT akan gagal jika kita mencoba melanggar constraint yang ada pada tabel (entah itu constraint NOT NULL, unique constraint, CHECK constraint, dan lain sebagainya).

Beberapa contoh:

```
CREATE TABLE t1 (i INT, j INT NOT NULL DEFAULT -1);

INSERT INTO t1 VALUES (1, 1);

INSERT INTO t1 VALUES (2, 2);

-- i dan j akan diisi nilai defaultnya yaitu NULL dan -1
INSERT INTO t1 VALUES (DEFAULT, DEFAULT);

-- j akan diisi nilai defaultnya, -1
INSERT INTO t1 (i) VALUES (3);

-- i dan j akan diisi nilai defaultnya
INSERT INTO t1 () VALUES ();
```

Setelah kelima perintah INSERT di atas, tabel akan berisi:

```
mysql> SELECT * FROM t1;
+------+----+
| i    | j  |
+------+----+
|    1 |  1 |
|    2 |  2 |
| NULL | -1 |
|    3 | -1 |
| NULL | -1 |
+------+----+
5 rows in set (0.07 sec)
```

Setiap kita tidak menyebutkan sebuah kolom dalam INSERT, atau menyebutkan nilai DEFAULT, maka sebuah kolom akan diisi dengan nilai defaultnya. INSERT dengan menyebutkan deretan nama kolom memiliki keuntungan yaitu saat sebuah tabel memiliki banyak kolom atau urutannya kita tidak hafal. Misalnya untuk tabel-tabel yang kompleks atau saat tabel telah di-ALTER TABLE beberapa kali sehingga urutannya tidak jelas lagi (tentu saja, sebetulnya kita selalu bisa melakukan DESCRIBE untuk melihat struktur tabel). Sementara INSERT tanpa menyebutkan deretan nama kolom memiliki keuntungan lebih pendek sintaksnya dibanding harus menyebutkan semua nama kolom dulu sebelum klausa VALUES.

Catatan: Standar SQL mendukung sintaks INSERT INTO ... DEFAULT VALUES untuk mengisi sebuah baris baru yang nilai kolomnya default semua. MySQL saat ini tidak mendukung sintaks tersebut, namun perintah tersebut bisa diganti dengan INSERT INTO ... VALUES (DEFAULT, DEFAULT, ...) atau dengan INSERT INTO ... () VALUES ().

Beberapa contoh lain untuk INSERT:

```
ALTER TABLE t1 ADD COLUMN k VARCHAR(32) NOT NULL;

-- error, karena sekarang tabel t1 memiliki 3 kolom: i, j, dan k
INSERT INTO t1 VALUES (1, 1);

-- tidak error, k akan diisi nilai defaultnya yaitu '' (string
kosong)
INSERT INTO t1 (i, j) VALUES (1, 1);
```

```
-- tidak error
INSERT INTO t1 VALUES (1, 1, 'satu');

-- mengisi dengan ekspresi. hasilnya: (2, 5, 'anak ayam')
INSERT INTO t1 VALUES (1+1, 2*2+1, CONCAT('anak', ' ', 'ayam'));

-- hasilnya: (NULL, -1, NULL), ingat pembahasan di Bab 3 mengenai
--   sifat NULL yang "menular"
INSERT INTO t1 (i, k) VALUES (1/NULL, CONCAT('gabung',' dengan',
NULL));

-- error, karena kolom j memiliki constraint NOT NULL
INSERT INTO t1 VALUES (
  1/NULL,
  NULL+1,
  CONCAT('gabung',' dengan', NULL)
);

ALTER TABLE t1 ADD COLUMN m INT UNIQUE;
INSERT INTO t1 (m) VALUES (1);

-- error, karena melanggar unique constraint untuk kolom m
INSERT INTO t1 (m) VALUES (1);

-- nilai default m adalah NULL
INSERT INTO t1 (m) VALUES (DEFAULT);

-- tidak error, karena kolom m dapat mengandung multiple NULL
INSERT INTO t1 () VALUES ();
```

Catatan untuk MySQL: MySQL mendukung INSERT banyak baris sekaligus dalam sebuah statement, dengan sintaks:

```
-- Resep 5-1-2: INSERT banyak baris sekaligus
INSERT INTO ... [(nama_kolom[, ...])] VALUES (...)[, (...), ...]
```

dengan kata lain, menyebutkan tupel nilai dipisahkan dengan koma. Namun ini tidak ada dalam standar SQL jadi perhatikan bahwa banyak database lain tidak mendukung hal ini.

Insert dan VARCHAR. Jika kita melakukan INSERT ke kolom VARCHAR, maka

string yang mengandung spasi di akhir (trailing blanks) akan otomatis dipangkas (trim) oleh MySQL . Kelakuan ini juga didapati di produk-produk database lain. Tipe data BLOB dan TEXT tidak memiliki kelakuan *autotrim* ini.

INSERT dan data di luar rentang. Jika kita memasukkan nilai yang di luar rentang tipe data, misalnya memasukkan nilai 1.000.000 ke kolom SMALLINT atau string 100 karakter ke kolom VARCHAR(64), maka MySQL tetap akan menerima nilainya dengan memotongnya sesuai nilai atau kapasitas tertinggi yang dimiliki tipe data kolom. Jadi pada kasus contoh ini, nilai 32767 akan dimasukkan ke kolom SMALLINT dan string 100 akan dipangkas menjadi hanya 64 karakter. Mulai versi 4.1, MySQL memberikan warning yang dapat Anda lihat dengan perintah SHOW WARNINGS. Database lain rata-rata berkelakuan lebih ketat yaitu menolak nilai-nilai invalid ini. Saya pribadi lebih suka dengan kelakuan yang lebih ketat ketimbang "main potong diam-diam" ala MySQL, jadi harapan saya mudah-mudahan di masa depan ada opsi di MySQL untuk memperketat hal ini.

INSERT IGNORE

```
-- Resep 5-1-3: Sintaks INSERT IGNORE
INSERT IGNORE INTO ...;
```

Opsi IGNORE untuk INSERT diperkenalkan oleh MySQL dan tidak dikenal di standar SQL. Opsi IGNORE akan membatalkan perintah INSERT yang bersangkutan jika ternyata penambahan baris melanggar unique constraint atau PK constraint. Misalnya, menyambung contoh sebelumnya:

```
INSERT INTO t1 (1, 1, 'satu', 1);
```

akan menghasilkan pesan kesalahan karena melanggar unique constraint pada kolom *m*, dikarenakan sudah ada baris yang nilai *m*-nya 1. Jika kita tambahkan opsi IGNORE:

```
INSERT IGNORE INTO t1 (1, 1, 'satu', 1);
```

maka perintah di atas akan berhasil tapi tidak menambahkan baris yang dimaksud. INSERT IGNORE biasa berguna untuk memastikan kita telah menambahkan baris tertentu; dengan INSERT IGNORE kita tidak perlu melakukan pengecekan dulu dengan perintah SELECT akan sebuah baris sudah ada atau belum.

Jika sebuah tabel tidak mengandung unique constraint atau PK, maka INSERT IGNORE menjadi ekivalen dengan INSERT biasa.

5.2 REPLACE INTO

```
-- Resep 5-2-1: Sintaks REPLACE INTO
REPLACE INTO nama_tabel [(nama_kolom      [, ...])]
                VALUES  (ekspresi|DEFAULT[, ...])
```

REPLACE INTO pada dasarnya sama dengan INSERT, namun memiliki kelakuan: jika penambahan baris baru melanggar unique constraint atau PK, maka baris sebelumnya akan diganti dengan baris baru. Jadi REPLACE INTO sifatnya berlawanan dengan INSERT IGNORE. INSERT IGNORE akan membiarkan baris lama yang sudah ada, REPLACE INTO akan mengganti dengan baris baru.

Catatan: REPLACE INTO tidak dikenal di standar SQL, namun SQL:2003 memperkenalkan perintah MERGE yang fungsinya sama dengan REPLACE INTO. Saat ini banyak database masih belum mendukung MERGE.

Contoh INSERT IGNORE vs REPLACE INTO:

```
CREATE TABLE t2 (i INT PRIMARY KEY, s VARCHAR(32) NOT NULL);
INSERT INTO t2 VALUES (1, 'Bandung');

-- gagal, melanggar PK
INSERT INTO t2 VALUES (1, 'Jakarta');

-- tidak akan melakukan apa-apa, tabel tetap berisi (1, 'Bandung')
INSERT IGNORE INTO t2 VALUES (1, 'Jakarta');

-- tabel kini akan berisi (1, 'Jakarta'); baris sebelumnya diganti
REPLACE INTO t2 VALUES (1, 'Jakarta');
```

Jika tidak ada unique constraint atau PK sama sekali pada tabel, maka REPLACE INTO ekivalen dengan perintah INSERT biasa.

5.3 Cara Lain Mengisi Tabel

Cara utama di SQL untuk mengisi baris baru ke dalam tabel adalah dengan perintah INSERT atau variannya (seperti MERGE atau REPLACE INTO). Perintah ini bisa dilakukan secara interaktif dari klien console mysql, secara programatik dari bahasa pemrograman, atau secara batch/noninteraktif dari sebuah file berisi perintah-perintah SQL yang di-feed ke program klien console (misalnya, file teks

berisi 1000 perintah INSERT untuk menambahkan 1000 baris ke dalam tabel). Namun INSERT bukan satu-satunya cara menambahkan baris baru.

Impor Dari CSV

Tiap database juga umumnya memiliki cara untuk mengimpor data dari file. File biasanya berupa file teks dengan format tab-separated atau comma-separated (CSV). Setiap baris tabel dinyatakan dalam sebuah baris teks dan tiap kolom dipisahkan dengan karakter tab atau koma. Cara ini biasanya lebih cepat daripada sekumpulan perintah INSERT dan memang ditujukan untuk *bulk loading* (mengisi tabel dengan banyak data sekaligus pada saat awal/inisialisasi tabel). Standar SQL saat ini tidak mengatur mengenai hal ini jadi tiap database punya perintahnya masing-masing. PostgreSQL misalnya, memiliki perintah COPY. MySQL sendiri memiliki LOAD DATA INFILE. Berikut ini sintaksnya:

```
-- Resep 5-3-1: Sintaks LOAD DATA INFILE
LOAD DATA [LOCAL] INFILE 'path_file'
    [REPLACE | IGNORE]
    INTO TABLE nama_tabel
    [FIELDS
        [TERMINATED BY '\t']
        [[OPTIONALLY] ENCLOSED BY '']
        [ESCAPED BY '\\' ]
    ]
    [LINES
        [STARTING BY '']
        [TERMINATED BY '\n']
    ]
    [IGNORE number LINES]
    [(nama_kolom,...)];
```

Perintah ini dapat mengimpor dari file teks secara cukup generik. Karakter pemisah antarbaris (record) dan antarkolom (field) dapat ditentukan sendiri.
Opsi LOCAL untuk memilih apakah *path_file* merupakan file yang ada di sisi klien atau sisi server. Jika LOCAL disebutkan, maka di sisi klien; sebaliknya jika tidak disebutkan berarti di sisi server database. Ingat bahwa MySQL defaultnya beroperasi sebagai server database yang dapat menerima koneksi dari komputer remote.

Opsi REPLACE atau IGNORE untuk memilih apakah LOAD DATA INFILE akan berkelakuan seperti perintah REPLACE INTO (mengganti baris lama) atau INSERT IGNORE (membiarkan baris lama). Jika tidak disebutkan, maka

kelakuannya akan seperti INSERT biasa (yakni, menghasilkan error/gagal jika terjadi pelanggaran unique constraint atau PK).

Opsi IGNORE ... LINES untuk melewati (skip) satu hingga beberapa baris di awal file (pada umumnya header).

File .CSV dapat dihasilkan oleh Excel dan banyak program lain, sehingga merupakan salah satu jalan yang sering ditempuh untuk mengimpor data dari berbagai program yang ada ke dalam database. Format CSV memisahkan antarfield dengan karakter koma (atau kadang-kadang titik koma) dan antarrecord dengan newline ("\n"). Sebuah field yang mungkin mengandung koma dapat dikutip dengan kutip ganda (""). Berikut ini perintah LOAD DATA INFILE untuk mengimpor file .CSV:

```
-- Resep 5-3-2: Sintaks LOAD DATA INFILE untuk mengimpor CSV
LOAD DATA [LOCAL] INFILE 'path_file'
    [REPLACE | IGNORE]
    INTO TABLE nama_tabel
    FIELDS
        TERMINATED BY ','
        OPTIONALLY] ENCLOSED BY '"'
        ESCAPED BY '\\'
    [IGNORE number LINES]
    [(nama_kolom,...)];
```

Dari Query (Perintah Select)

Perintah CREATE TABLE dan INSERT sebetulnya menerima klausa SELECT. Perintah SELECT adalah query yang menghasilkan satu atau lebih baris, sehingga kita dapat langsung mengisi sebuah tabel dengan baris-baris ini.

```
-- Resep 5-3-3: Mengisi tabel dengan output query
CREATE TABLE ... ( ... ) ...
  perintah_select;

INSERT INTO ... [(nama_kolom, ...)]
  perintah_select;
```

Di mana perintah_select adalah perintah SELECT dengan segala variasi sintaks dan fiturnya. (Kita akan mempelajari SELECT dengan lebih mendalam pada bab-bab berikutnya setelah ini.)

CREATE TABLE yang menerima query sebetulnya tidak perlu mendefinisikan kolom. Definisi nama kolom dan jenis datanya akan dideduksi dari hasil SELECT. Namun jika kita ingin tipe data yang agak berbeda, atau ingin menambahkan constraint atau indeks atau hal lainnya, maka kita tetap perlu mendefinisikannya. Catatan: nama kolom di definisi CREATE TABLE dengan nama kolom di query haruslah sama, jika tidak MySQL akan membuatkan dua buah kolom berbeda.

Catatan: CREATE TABLE yang menerima query SQL dikenal di standar SQL:1999 dan SQL:2003. Sintaksnya sedikit berbeda yaitu CREATE TABLE ... AS SELECT .... (ada token AS).

Contoh:

```
CREATE TABLE t1 (i INT, j VARCHAR(32));
INSERT INTO t1 VALUES (1, 'satu');
INSERT INTO t1 VALUES (2, 'dua');
INSERT INTO t1 VALUES (3, 'tiga');

CREATE TABLE kuadrat1 (n SMALLINT NOT NULL, k INT NOT NULL)
  SELECT i, i*i FROM t1;
CREATE TABLE kuadrat2 (n SMALLINT NOT NULL, k INT NOT NULL)
  SELECT i AS n, i*i AS k FROM t1;
```

Hasilnya, tabel *kuadrat1* (perhatikan empat buah kolom jadi terbentuk, dikarenakan tidak menyamakan nama kolom):

```
mysql> SELECT * FROM kuadrat;
+---+---+------+------+
| n | k | i    | i*i  |
+---+---+------+------+
| 0 | 0 |    1 |    1 |
| 0 | 0 |    2 |    4 |
| 0 | 0 |    3 |    9 |
+---+---+------+------+
3 rows in set (0.00 sec)
```

Tabel *kuadrat2* (sesuai yang diinginkan):

```
mysql> SELECT * FROM kuadrat2;
+---+---+
| n | k |
```

```
+---+---+
| 1 | 1 |
| 2 | 4 |
| 3 | 9 |
+---+---+
3 rows in set (0.00 sec)
```

Contoh lain:

```
CREATE TABLE ulang
  SELECT CONCAT(i,'-',i) AS i, CONCAT(j,'-','j') AS j FROM t1;
```

Hasilnya, MySQL akan berusaha membuatkan kolom dengan jenis data dan panjang yang cocok (menurutnya):

```
mysql> SELECT * FROM ulang;
+------+-----------+
| i    | j         |
+------+-----------+
| 1-1  | satu-satu |
| 2-2  | dua-dua   |
| 3-3  | tiga-tiga |
+------+-----------+
3 rows in set (0.00 sec)

mysql> DESCRIBE ulang;
+-------+------------+------+-----+---------+-------+
| Field | Type       | Null | Key | Default | Extra |
+-------+------------+------+-----+---------+-------+
| i     | binary(23) | YES  |     | NULL    |       |
| j     | char(65)   | YES  |     | NULL    |       |
+-------+------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

Di sini bisa dilihat bahwa kolom *i* didefinisikan sebagai string bertipe BINARY (= CHAR BINARY) dengan panjang 23, ini karena kolom INT dapat memiliki panjang hingga sebelas karakter (nilai maksimum 2 milyar memiliki 10 digit ditambah tanda minus jika negatif). Kita melakukan CONCAT() untuk menggabung secara string dua buah kolom *i* dengan tanda minus, sehingga total panjang maksimum = 11 + 1 + 11 = 23. Demikian juga *j* pada tabel *t1* memiliki panjang 32 sehingga pada tabel *ulang* menjadi 32 + 1 + 32 = 65. Jika Anda tidak suka pada hasil kesimpulan

MySQL dalam menentukan jenis dan panjang kolom, silakan definisikan sendiri di CREATE TABLE.

5.4 UPDATE

```
-- Resep 5-4-1: Sintaks UPDATE
UPDATE [IGNORE] nama_tabel
  SET nama_kolom=ekspresi[, ...]
  [WHERE kondisi]
  [ORDER BY ...]
  LIMIT jumlah_baris;
```

Perintah UPDATE digunakan untuk mengupdate nilai kolom untuk baris-baris yang sudah ada pada tabel. Contoh:

```
CREATE TABLE propinsi (
  id INT PRIMARY KEY,
  nama VARCHAR(100) NOT NULL,
  ibukota VARCHAR(32)
);
INSERT INTO propinsi VALUES (1, 'jabar', 'bandung');
INSERT INTO propinsi VALUES (2, 'irjabar', NULL);
  -- terakhir dengar, masih diperdebatkan antara Sorong dan
Manokwari
INSERT INTO propinsi VALUES (3, 'gorontalo', 'gorontalo');
INSERT INTO propinsi VALUES (4, 'jakarta', 'jakarta');

-- 1. berhasil, sekarang jabar memiliki id 10, irjabar 20,
gorontalo
--    30, dan seterusnya.
UPDATE propinsi SET id=id*10;

-- 2. kembalikan pada nilai semula
UPDATE propinsi SET id=id/10;

-- 3. gagal, karena saat ingin mengubah id jabar jadi 2, bentrok
dengan
--    id milik irjabar yang 2 pula, sehingga tak ada id yang
berubah.
UPDATE propinsi SET id=id*2;
```

```
-- 4. berhasil, karena kita melakukan pengubahan dari yang
terbesar
--    (jakarta, id=4; disusul gorontalo, id=3, dan seterusnya.)
UPDATE propinsi SET id=id*2 ORDER BY id DESC;

-- 5. kembalikan ke nilai semula (urutan pengubahan kita balik
agar
--    tidak bentrok)
UPDATE propinsi SET id=id/2 ORDER BY id;

-- 6. hanya mengubah sebagian baris, yakni yang nama propinsi sama
--    dengan nama ibukotanya. yang terubah adalah 'jakarta' dan
--    'gorontalo'
UPDATE propinsi
  SET
    nama=CONCAT('propinsi ', nama),
    ibukota=CONCAT('kota ', ibukota)
  WHERE nama=ibukota;

-- hasil no 6:
mysql> SELECT * FROM propinsi;
+----+-------------------+---------------+
| id | nama              | ibukota       |
+----+-------------------+---------------+
|  1 | jabar             | bandung       |
|  2 | irjabar           | NULL          |
|  3 | propinsi gorontalo | kota gorontalo |
|  4 | propinsi jakarta  | kota jakarta  |
+----+-------------------+---------------+
4 rows in set (0.01 sec)

-- 7. sama seperti nomor 6, tapi kita menggunakan klausa LIMIT
untuk
--    membatasi jumlah baris yang diubah. Dalam hal ini yang
berubah
--    hanya 'gorontalo'.
UPDATE propinsi
  SET
    nama=CONCAT('propinsi ', nama),
    ibukota=CONCAT('kota ', ibukota)
```

```
  WHERE nama=ibukota
  ORDER BY nama
  LIMIT 1;
```

Seperti diperlihatkan pada contoh-contoh di atas, kita dapat menginstruksikan UPDATE agar mengubah dengan urutan tertentu untuk mencegah melanggar constraint. Dan juga membatasi hanya melakukan pengubahan sekian baris saja. Kedua fasilitas ini merupakan fitur spesifik MySQL. Di standar SQL, klausa ORDER BY dan LIMIT untuk perintah UPDATE tidak dikenal. Demikian juga opsi IGNORE, yaitu untuk membuat perintah UPDATE tidak berbuat apa-apa jika unique key atau primary key terlanggar.

UPDATE berkelakuan atomik. Jika misalnya ada 10 baris yang bisa diubah dalam sebuah perintah UPDATE, lalu pengubahan baris ke-10 melanggar constraint, maka kesembilan pengubahan sebelumnya akan dibatalkan. Sehingga perintah UPDATE tidak mengubah baris-baris pada tabel sama sekali.

UPDATE juga secara default akan mengubah *semua* baris pada tabel, kecuali jika Anda batasi dengan WHERE, yaitu untuk memilih hanya baris-baris yang memenuhi kondisi pada klausa WHERE tersebut. Jika tidak ada baris yang cocok, maka UPDATE tidak akan mengubah satu barispun pada tabel.

Lupa menyebutkan klausa WHERE adalah salah satu kesalahan pemula yang sering terjadi, sehingga terjadi kecelakaan terubahnya semua baris lain yang tidak diinginkan. Contoh, Anda menulis:

```
UPDATE propinsi SET ibukota='manokwari';
```

padahal maksud Anda adalah:

```
UPDATE propinsi SET ibukota='manokwari' WHERE nama='irjabar';
```

akibatnya, semua propinsi akan diset ibukotanya menjadi Manokwari. Ups! Jadi berhati-hatilah dalam melakukan UPDATE.

5.5 Menghapus Data Dari Tabel

DELETE

```
-- Resep 5-5-1: Sintaks DELETE
DELETE [IGNORE] FROM nama_tabel
```

```
WHERE kondisi
ORDER BY ...
LIMIT jumlah_baris;
```

Perintah DELETE menghapus nol atau lebih baris pada sebuah tabel. Sama seperti pada UPDATE, klausa ORDER BY, LIMIT, dan IGNORE merupakan fitur spesifik MySQL dan tidak ada pada standar SQL.

TRUNCATE

```
-- Resep 5-5-2: Sintaks TRUNCATE
TRUNCATE nama_tabel;
```

Perintah ini menghapus semua baris pada tabel, alias mengosongkan tabel.

DROP TABLE

Jika Anda sudah tidak membutuhkan sebuah tabel, bisa juga Anda hapus tabelnya itu sendiri. (Lihat Resep 4-7-1).

5.6 Transaksi

Kadang-kadang dalam sebuah proses, kita menginginkan dua atau lebih perubahan pada satu atau lebih tabel database terjadi secara serempak. Atau gagal serempak, sehingga tidak pernah setengah-setengah. Dengan kata lain, kita ingin sekelompok perintah tersebut terjadi secara *atomik*. Contoh yang paling sering dipakai (dan akan juga saya pakai di sini) adalah transfer bank dari satu rekening ke rekening lainnya. Dalam proses ini, account sumber harus didebit (dikurangi jumlahnya) sementara account tujuan dikredit (ditambah jumlahnya dengan nilai yang sama). Kedua perintah perubahan ini tentunya harus atomik, tidak bisa setengah-setengah. Jika terputus di tengah-tengah, maka entah bank atau si pemilik account sumber yang rugi (bergantung pada debit dulu atau kredit dulu yang dilakukan). Contoh lain proses yang harus atomik misalnya pembelian barang menggunakan kartu kredit di Internet (antara charging kartu kredit di database bank dan pencatatan transaksi di database merchant/payment gateway), penarikan uang di ATM, posting transaksi di jurnal akuntansi, penambahan atau penghapusan order (ke tabel *pesanan* dan *detail_pesanan*), dan lain sebagainya.

Selain atomitas, hal lain yang penting adalah isolasi. Jika dua buah proses mengubah sebuah record dalam waktu bersamaan, perlu adanya resolusi siapa yang berhasil dan siapa yang gagal. Database tidak boleh mengatakan kedua proses berhasil, padahal sebetulnya hasil dari proses 1 ditimpa oleh proses 2. Melainkan,

database harus melaporkan misalnya pada proses 2 bahwa terjadi konflik dan pengubahan dibatalkan/ditolak.

Kedua sifat di atas, ditambah lagi konsistensi dan durabilitas, merupakan sesuatu yang ditawarkan oleh transaksi. Transaksi memiliki sifat ACID: atomic, consistent, isolated, dan durable (sekali transaksi selesai, pengubahan diflush ke disk dan menjadi permanen). Dalam kenyataan sehari-hari, konsep transaksi ini sering sekali diperlukan. Dan konsep transaksi merupakan sesuatu yang telah dikenal lama di komunitas database jauh sebelum model relasional maupun SQL.

Intinya adalah, kelompok proses-proses pengubahan yang kita lakukan perlu dibungkus dengan transaksi (atau perlu menjadi transaksi-transaksi) agar pengubahan kita bersifat konsisten, atomik, terisolasi dari bentrokan dengan proses/transaksi lain, dan bersifat permanen saat telah sukses. Jika kita tidak menggunakan transaksi, maka bisa saja jika terjadi crash hardware atau kesalahan coding, database kita menjadi tidak konsisten karena ada perubahan yang terekam setengah-setengah.

MySQL mendukung transaksi, namun saat ini baru untuk engine BerkeleyDB dan InnoDB. Jenis engine MyISAM saat ini belum ACID-compliant.

Untuk membungkus kelompok perintah SQL dengan transaksi, gunakan perintah BEGIN [WORK] dan akhiri dengan COMMIT atau ROLLBACK.

```
-- Resep 5-6-1: Membungkus sekelompok perintah SQL dengan
transaksi
BEGIN [WORK];
perintah SQL...
COMMIT | ROLLBACK;
```

Perintah SQL berupa perintah-perintah DML (INSERT, UPDATE, SELECT, DELETE). Pada database seperti PostgreSQL, perintah-perintah DDL seperti CREATE TABLE dan ALTER TABLE bersifat transaksional dan dapat dibungkus transaksi.

Setelah kita selesai dengan berbagai perintah SQL dalam sebuah transaksi, kita dapat memberikan perintah COMMIT untuk menyatakan menyetujui transaksi tersebut. Semua perintah SQL akan menjadi berefek permanen dan transaksi selesai. Sebaliknya, jika kita memberikan perintah ROLLBACK, kita menyatakan ingin membatalkan transaksi yang bersangkutan. Semua perubahan yang tadi dilakukan akan dibatalkan efeknya. COMMIT tentu saja bisa gagal, misalnya

database crash atau disk penuh. Atau di tengah-tengah perintah SQL terjadi kesalahan dan transaksi otomatis dibatalkan oleh server database. Dalam semua kasus, sifat transaksi tetap terjaga yaitu atomik (semua-gagal atau semua-berhasil), terisolasi, konsisten, dan permanen.

Contoh:

```
CREATE TABLE t1 (i INT) ENGINE=innodb;
BEGIN;
INSERT INTO t1 VALUES (1);
INSERT INTO t1 VALUES (2);

-- 1. lihat hasil sementara
mysql> SELECT * FROM t1;
+------+
| i    |
+------+
|    1 |
|    2 |
+------+
2 rows in set (0.00 sec)

– 2. batalkan transaksi
ROLLBACK;

– 3. lihat hasil akhir
mysql> SELECT * FROM t1;
Empty set (0.00 sec)
```

Pada tahap 1, transaksi belum dikomit jadi seandainya mesin mati atau proses MySQL dibunuh, ketika start lagi maka tabel *t1* akan tetap kosong karena transaksi otomatis dibatalkan (dirollback). Dengan kata lain, sebelum komit perubahan belum bersifat permanen.

Pada tahap 2, kita membatalkan transaksi, sehingga tabel kembali dalam kondisi seperti saat kita belum melakukan transaksi. Jika misalnya tabel sudah berisi baris lalu kita hapus dalam transaksi dan transaksi dibatalkan, maka baris yang tadi dihapus akan muncul kembali, karena perubahan belum permanen.

InnoDB beroperasi pada level isolasi tertinggi yaitu serializable (ada 4 level isolasi yang diatur dalam standar SQL, dari yang rendah/tidak terisolasi hingga tinggi/

terisolasi total). Pada level isolasi transaksi ini, perubahan yang terjadi selama transaksi tidaklah terlihat oleh transaksi lain selama si transaksi yang melihat belum melakukan komit. Dengan kata lain, saat sebuah transaksi mulai berjalan, seolah-olah dunia berhenti. Semua perubahan yang terjadi oleh transaksi lain tidak akan terlihat. Misalnya, pada contoh sebelumnya, cobalah buka klien console mysql kedua setelah Anda berada di tahap 1. Lalu berikan perintah SELECT * FROM t1. Anda akan melihat bahwa tabel masih kosong, padahal klien console yang pertama telah memberikan perintah INSERT. Demikian pula jika tahap 2 pada klien console pertama diganti dengan COMMIT, si klien console kedua belum juga akan melihat perubahan oleh transaksi pertama. Barulah setelah klien console kedua melakukan COMMIT (atau ROLLBACK, tentunya) semua perubahan yang terjadi kini nampak.



5.6 Transaksi